REMARKS

Claims 1-24 are pending in the present application. Claims 4, 6, 8, 15, 17, and 19 are canceled. Claims 1, 5, 7, 11, 12, 16, 18, 22, 23, and 24 are amended. Claims 25-30 are added. Reconsideration of the claims is respectfully requested.

I.   35 U.S.C. § 102, Anticipation

The Office Action rejects claims 1-3, 5, 7, 9-14, 16, 18, 20, 21, 23, and 24 under 35 U.S.C. § 102 as being anticipated by *Walker* (U.S. Patent No. 6,138,171). This rejection is respectfully traversed.

Walker teaches a generic software state machine for implementing a software state machine in an object oriented environment. Figure 4 of *Walker* is as follows:
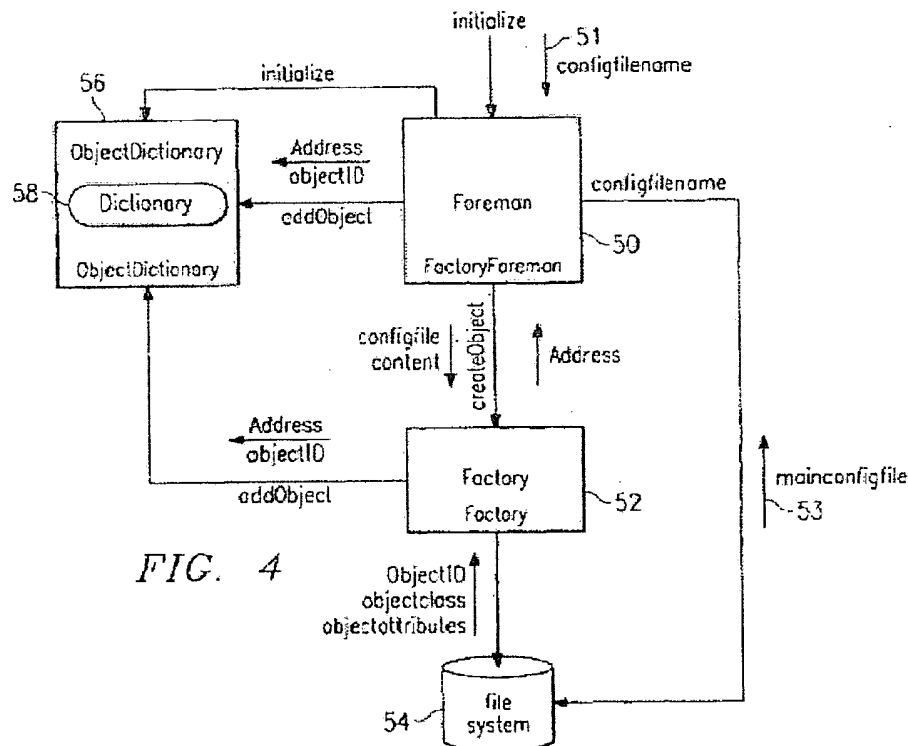

FIG. 4

A foreman object is provided a specific configuration file that specifies and defines a set of top level objects that need to be created by a factory object. The foreman assumes all object building responsibilities and stores an address for each object that is created in an object dictionary. See *Walker*, col. 8, lines 8-34.

Furthermore, *Walker* states:

> Accordingly, to achieve the functionality of the software state machine an exemplary set of class hierarchies has been designed. Some of these class hierarchies have been briefly described above and summarized below:
>
> > All received messages are derived from a Message class which hides all application detail from the software state machine.
> >
> > A Thread class is created which defines a pure virtual function for translating OSMessage instances into compatible Message instances.
> >
> > An FsmMap class supports individual finite state machine instances as well as collections of finite state machine instances.
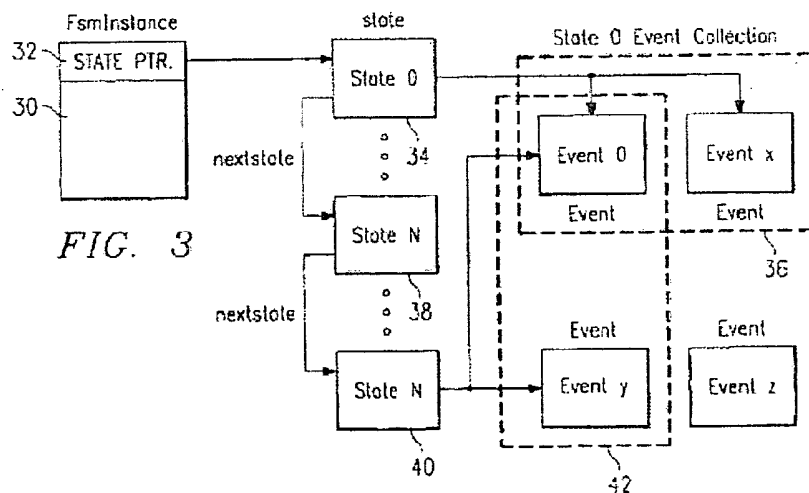> >
> > An FsmEntity class is the abstract interface between the software state machine and call blocks.
> >
> > A State class defines the abstract behavior for entering, exiting a state and processing an event.
> >
> > New state classes, dictionary state and configurable state, support logical state changes. The user is responsible for defining the set of logical state variables and the mapping within each state of logical state to physical state identifier.
> >
> > Event class hierarchy supports processing of a specific event for a state.

Walker, col. 10, line 48, to col. 11, line 5. In *Walker*, an instance of a finite state machine (FSM) object (FSMInstance) processes messages. Figure 3 of *Walker* is as follows:

FIG. 3

Page 9 of 17
Cook – 10/047,801

Each FSM instance is associated with a current state and the address of a state that specifies the processing of messages. The FSM instance object calls the current state's exit member function and repeatedly calls the next state's enter function until the next state's enter function does not return a state identifier. Instances of an event state object contain a state dictionary of the events that are defined for the state. Received message identifiers are translated into the address of one or more target events. The event state object then passes the state machine message and the FSM instance object to the event object. An event instance returns a pointer to a logical state instance when a state change is required. The event state object then translates the logical state returned by the event object into a state identifier, which is returned to the caller. See *Walker*, col. 9, line 32, to col. 10, line 3.

The factory object creates all objects by retrieving the object identifiers, object classes, and optional object attributes of the object classes enumerated in the configuration file. The factory then provides the addresses of the created objects to the foreman. See *Walker*, col. 8, lines 35-51. Thus, the configuration file must define a message class, a thread class, an FSM map class, an FSM entity class, a state class, a new state class, and an event class hierarchy. The factory of *Walker* merely creates the objects according to these definitions. The FSM instance object of *Walker* calls functions in these objects and the objects themselves process messages, pass object identifiers, point to a logical state instance, and so forth.

While the applied reference does teach creating and executing a software state machine, the claimed invention creates a software state machine in a different manner. Claim 1, for example, recites:

1. A method for creating a software state machine, comprising:
providing a state machine object; and
providing an initializer object, wherein the initializer object defines states, actions, and conditions for a software state machine,
 wherein the state machine object is configured to use the initializer object to create a table object,
 wherein the state machine object is configured to create an array of state variables,
 wherein the table object is configured to create an array of state transition objects based on the array of state variables and return the array of state transition objects to the state machine object, and

wherein the state machine object is configured to execute the
software state machine using the array of state transition objects.

Walker does not teach or suggest a finite state machine object that uses an initializer
object to create a table object, wherein the table object that creates an array of state
transition objects and returns the array of state transition objects to the finite state
machine.

In *Walker*, all high level objects for the finite state machine are created by the
foreman object and the factory object. The foreman object of *Walker* is not a sate
machine object that executes the finite state machine as alleged in the Office Action. In
fact, the foreman object and factory object of *Walker* create the high level objects of the
software state machine. *Walker* further teaches that the high level objects of the software
state machine execute the software state machine. The FSM instance object of *Walker*
does not create a table object, as recited in the claims.

Furthermore, with respect to claims 4-6, the Office Action alleges that *Walker*
teaches that the foreman object, which is not a state machine object as claimed, is
configured to create a table object at col. 8, lines 22-65. The Office Action alleges that
the object dictionary of *Walker* is equivalent to the claimed table object. Applicant
respectfully disagrees. The cited portion of *Walker* states:

> Foreman 50 assumes all of the object building
> responsibilities. It is responsible for constructing the system
> in preparation for start-up. Foreman stores the ObjectID
> and object address of each created object in the
> objectDictionary. The factory object may reference a
> hierarchy of objects in the creation of the requested object.
> The Foreman constructs the system in two steps:
> 1. All objects are created and stored in the
>   objectDictionary. At this stage, objects are linked
>   by ObjectID.
> 2. The objectDictionary initialize method is called.
> The dictionary iterates through all objects executing
> their initialize method. All objects should translate
> ObjectID references into physical object addresses.
> Factory 52 contains a set of Factory objects. Each
> Factory object possess the knowledge to construct an
> instance of a particular class of objects. Referring also to
> FIG. 5, Foreman 50, Factory 52, and other objects are part
> of a dynamic object creator 70. One instance of a Factory

object is created in the Factory object constructor for each
type of object that can be dynamically constructed.
Dynamic objects that may be created by dynamic object
creator 70 includes collection objects 72, thread objects 74,
queue objects 76, timer objects 78, state objects 80, and
message objects 82, for example. Application programs 90
are built on top of the dynamic objects 72-82 constructed
by dynamic object creator 70 and utilize these dynamic
objects to perform needed functions. Therefore, factory 52
may be unique to each application to create different sets of
dynamic objects suited to each application. The dynamic
objects of the application program are generally created in
two steps.

Referring again to FIG. 4, Factory 52 creates all objects
by retrieving the object identifiers (ObjectID), object
classes, and optional object attributes of the object classes
enumerated in configuration file 53. Factory 52 then
provides the addresses of the created objects to Foreman 50
and the addresses and object identifiers of the created
objects to an ObjectDictionary 56, which maintains a table
or database for storing them. A dictionary object 58 within
ObjectDictionary 56 is then instructed to initialize all
objects stored in ObjectDictionary 56 via its initialize
member function. Dictionary 58 iterates through all of its
stored objects calling their member functions. References
between the created objects are then cross linked.
ObjectDictionary 56 is a static object that is visible to all
dynamically created objects.

Walker, col. 8, lines 8-22. Neither the cited portion nor any other portion of *Walker*
teaches or suggest that the foreman object creates the object dictionary. Also, the object
dictionary stores objects created by the foreman and the factory. The object dictionary of
Walker does not create an array of state transition objects and return the array of state
transition objects to the state machine object, as recited in claim 1, for example.
Therefore, the object dictionary of *Walker* is not equivalent to the claimed table object.

The applied reference does not teach or suggest each and every claim limitation;
therefore, *Walker* does not anticipate claim 1. Independent claims 12 and 23 recite
subject matter addressed above with respect to claim 1 and are allowable for similar
reasons. Since claims 2, 3, 5, 7, 9, 10, 13, 14, 16, 18, 20, and 21 depend from claims 1
and 12, the same distinctions between *Walker* and the invention recited in claims 1 and

12 apply for these claims. Additionally, claims 2, 3, 5, 7, 9, 10, 13, 14, 16, 18, 20, and 21 recite other additional combinations of features not suggested by the reference.

More particularly, claim 2 recites that the state machine object includes an object constructor method and claim 3 recites that the object constructor method is configured to create an instance of the initializer object. The Office Action alleges that *Walker* teaches these features at col. 8, lines 35-45. Applicant respectfully disagrees. The cited passage, reproduced above, does not make any mention of a state machine object including an object constructor method that creates an instance of an initializer object. Furthermore, *Walker* states:

> Foreman 50 retrieves MainConfig file 53 from a file system
> or database 54 and passes the contents of the configuration
> file to Factory 52.

Walker, col. 8, lines 16-19. Thus, *Walker* clearly teaches a pre-existing configuration file in a file system or database. *Walker* does not teach or suggest a state machine object that includes an object constructor method, wherein the object constructor method is configured to create an instance of an initializer object, as recited in claims 2 and 3, for example. Claims 13 and 14 recite subject matter addressed above with respect to claims 2 and 3 and are allowable for similar reasons.

With respect to claim 5, the Office Action alleges that *Walker* teaches an initializer object including a table element array creation method, wherein the state machine object is configured to call the table element array creation method to create the table object at col. 8, lines 22-65. With respect to claim 7, the Office Action alleges that Walker teaches an initializer object including a table variable array creation method, wherein the state machine object is configured to call the table variable array creation method to create an array of state variables at col. 8, lines 22-65. The cited passage, reproduced above, does not make any mention of an initializer object including a table element array creation method that creates a table object. In fact, *Walker* states the following:

> Constructed in this manner, an application program can
> be modified quickly by changing the configuration files
> without recompiling and relinking the code. Configuration
> files may be written in TCL, however a script language or

even ASCII may be used to specify the list of configuration
file names that stores object specifications.

Thus, *Walker* teaches that a configuration file merely stores object specifications or a list
of configuration file names. There is no teaching in any portion of *Walker* that a
configuration file may include a method that may be called to create a table object or an
array of state variables, as recited in claims 5 and 7, for example. Claims 16 and 18
recite subject matter addressed above with respect to claims 5 and 7 and are allowable for
similar reasons.

Still further, claim 11, as amended, recites:

11.   A method for creating software state machines, comprising:
providing a state machine object;
providing an initializer object, wherein the initializer object defines
a plurality of states, one or more actions, one or more inputs, one or more
conditions, one or more events, and one or more triggers for a software
state machine, wherein the state machine object is configured to use the
initializer object to create an array of state transition objects, wherein each
state transition object in the array of state transition objects defines at least
one of the one or more conditions that causes a given state transition in the
software state machine, and wherein each condition is a Boolean
expression formed from at least one of the one or more inputs; and
responsive to occurrence of a trigger, evaluating the one or more
inputs, computing the one or more conditions, and determining a next state
based on the array of state transition objects.

Walker does not teach or suggest an initializer object that defines a plurality of
conditions, wherein the state machine object is configured to use the initializer object to
create an array of state transition objects, wherein each state transition object in the array
of state transition objects defines at least one of the one or more conditions that causes a
given state transition in the software state machine, and wherein each condition is a
Boolean expression formed from at least one of the one or more inputs, as recited in
claim 11, for example. As described above, the state machine of *Walker* is executed by a
FSM instance object pointing to a current state. The FSM instance object of *Walker* then
calls a function of the current state and the next state to determine whether a state change
occurs. However, *Walker* does not teach an array of state transition objects defining
conditions that are Boolean expressions formed form inputs.

The applied reference does not teach each and every claim limitation; therefore, *Walker* does not anticipate claim 11. Independent claim 24 recites subject matter addressed above with respect to claim 11 and is allowable for the same reasons. Since newly added claims 25-27 and 28-30 depend from claims 11 and 24, the same distinctions between *Walker* and the invention recited in claims 11 and 24 apply for these claims. Additionally, claims 25-27 and 28-30 recite other additional combinations of features not suggested by the reference.

Therefore, Applicant respectfully requests withdrawal of the rejection of claims 1-21, 23, and 24 under 35 U.S.C. § 102.

Furthermore, *Walker* does not teach, suggest, or give any incentive to make the needed changes to reach the presently claimed invention. Absent the Office Action pointing out some teaching or incentive to implement Walker to have a state machine object create a table object that creates an array of state transition objects and return the array of state transition objects to the state machine object for execution of the state machine, one of ordinary skill in the art would not be led to modify *Walker* to reach the present invention when the reference is examined as a whole. Absent some teaching, suggestion, or incentive to modify *Walker* in this manner, the presently claimed invention can be reached only through improper use of hindsight using the Applicant's disclosure as a template to make the necessary changes to reach the claimed invention.

II.   35 U.S.C. § 103. Obviousness

The Office Action rejects claim 22 under 35 U.S.C. § 103 as being unpatentable over *Walker* (U.S. Patent No. 6,138,171). This rejection is respectfully traversed.

Claim 22 recites subject matter addressed above with respect to claims 1-3, 5, and 7 and is allowable for similar reasons. More particularly, *Walker* does not teach or suggest a finite state machine object that uses an initializer object to create a table object, wherein the table object that creates an array of state transition objects and returns the array of state transition objects to the finite state machine. Furthermore, *Walker* clearly teaches a pre-existing configuration file in a file system or database. *Walker* does not teach or suggest a state machine object that includes an object constructor method, wherein the object constructor method is configured to create an instance of an initializer

object. Still further, *Walker* teaches that a configuration file merely stores object specifications or a list of configuration file names. There is no teaching in any portion of *Walker* that a configuration file may include a method that may be called to create a table object or an array of state variables.

While virtual machines may be generally well known for emulating hardware paradigms, a virtual machine does not cure the deficiencies of *Walker*. Furthermore, there is no teaching in the prior art that would lead a person of ordinary skill in the art to modify the specific teaches of Walker such that a virtual machine would create an instance of the state machine object including an object constructor method, wherein the object constructor method creates an instance of the initializer object and uses the instance of the initializer object to create a table object and an array of state variables, wherein the table object includes a state array creation method and the object constructor method calls the state array creation method to create an array of state transition object, and wherein the instance of the state machine object uses the array of state transition objects to execute a software state machine, as recited in claim 22.

The prior art, when considered as a whole, fails to teach each and every claim limitation and would not lead a person of ordinary skill in the art to make the modifications necessary to arrive at the claimed invention. Therefore, *Walker* does not render claim 22 obvious. Therefore, Applicant respectfully requests withdrawal of the rejection of claim 22 under 35 U.S.C. § 103.
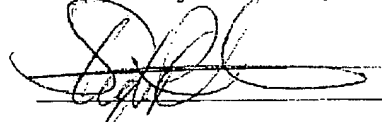
III.   Conclusion

It is respectfully urged that the subject application is patentable over the prior art of record and is now in condition for allowance.

The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: February 18, 2005

Respectfully submitted,

Stephen R. Tkacs
Reg. No. 46,430
YEE & ASSOCIATES, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Agent for Applicant

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

❏ BLACK BORDERS

❏ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES

☑ FADED TEXT OR DRAWING

❏ BLURRED OR ILLEGIBLE TEXT OR DRAWING

❏ SKEWED/SLANTED IMAGES

❏ COLOR OR BLACK AND WHITE PHOTOGRAPHS

❏ GRAY SCALE DOCUMENTS

❏ LINES OR MARKS ON ORIGINAL DOCUMENT

❏ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY

❏ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.